PARTIAL PAGE OUTPUT CACHING

FIELD OF THE INVENTION

[02] This invention relates in general to a method for providing partial page output caching for a networked computer system, and more particularly to a method and apparatus for providing server controlled data caching for portions of dynamically-generated web pages.

BACKGROUND OF THE INVENTION

[03] Over the last several years, the use of the Internet by individuals and business entities has increased significantly as the Internet has become established as a mechanism to disseminate information. The Internet presents information to a user using a web browser that is located on the user's computer. The web browser retrieves and displays web pages from various web servers connected to the Internet. The increase in use of the Internet has caused web servers to provide large numbers of users with the same page as the users request access to the same web pages or at least pages having portions that are the same.

[04] Initially, the web browsers, web servers, and intermediate proxy servers used data caching techniques to assist in shortening the response time when web pages are requested. These caching techniques stored a static version of a web page into data cache memory blocks at any number of locations between the end user's computer and

the web server that generated a response to a web page request. Web browsers typically have a local cache to hold temporary files retrieved while the browser is used. When a new web page is requested, the browser may check to see if the requested page exists in the local cache. If the page is in the cache, the web browser may retrieve the requested page from the cache and thus eliminate the need to request information from the web server. Because the cache is typically located locally upon a hard drive of a user's computer, no Internet communication is needed. This response is typically much quicker than sending a request over the Internet. In addition, the use of a cached page eliminates a subsequent web server hit, thereby reducing the processing requirements of the web servers.

[05] Proxy servers are servers located between two portions of a networked computing system. Typically, all of the users are attached to a network located on one side of a proxy server and the web servers providing the requested pages are located upon the other side of the proxy server. In this architecture, the user sends a request for a web page to the proxy server, which in turns sends a request for a web page to the appropriate web server. The web server responds with a web page to the proxy server. The proxy server then forwards the web page to the requesting user.

[06] The proxy servers occasionally possess local cache memory to hold web pages that have been previously requested by users. Similarly, the proxy server may check for the storage of a requested page within its cache before sending a request to the web server. Again, if the page is found, the cached version may be used to eliminate the web server request.

[07] The above uses of cache memory blocks have several deficiencies that diminish their effectiveness. First, these cache blocks store the complete version of a web page when it was last sent from a web server. As such, these cache memory blocks will hold only static web pages. At present, a significant amount of the web pages being requested by users include dynamically-generated content. As such, the web page

requested by a first user may not contain all of the same information as the web page requested by a second user. In this circumstance, each web page will need to be considered a unique and different web page. As a result, the benefits of data caching will not be obtained even though most of the data on the different web pages may be identical.

[08] U.S. Patent Application No. 09/570,071, filed on May 12, 2000, entitled, "Output Caching Module of an HTTP Pipeline and assigned to the same assignee as the present application, discloses a server caching an output page. When a dynamically-changing web page is requested by a user at a client computer and the web page is available in the web server's output cache, instead of regenerating the web page for output to a client computer, the server retrieves the web page from the output cache and sends the web page to the client computer.

[09] A disadvantage to the solution described in U.S. Patent Application No. 09/570,071 is that when portions of a web page are the same, but other portions differ, such web pages will be considered to be different web pages and therefore, could not make use of output page caching.

BRIEF SUMMARY OF THE INVENTION

[10] The present invention relates to a method for providing partial page output caching. The method provides for output caching a portion of a web page, thereby allowing the cached portion to be used in web pages for a predetermined period of time without the need to regenerate that portion of the web page.

[11] In an embodiment of the invention, a server computing system receives a request for information from a client computer system. The server computing system creates, in response to the request, a page having portions. When an output cache contains a portion of the page, the portion of the page contained in the output cache is retrieved from the output cache. When the output cache does not contain a portion of the page,

the portion of the page not contained in the output cache is retrieved from another source. The complete contents of the page are then sent to the client computing system.

[12] Another embodiment of the invention includes a machine-readable medium having instructions recorded thereon, such that when the instructions are read and executed by a processor in a computing system connected to a network, the computer is caused to function as a server computing system. The server computing system is then configured to create, in response to a request for information from a client computer system, a page having portions. When an output cache contains a portion of the page, the portion of the page contained in the output cache is retrieved from the output cache. When the output cache does not contain a portion of the page, the portion of the page not contained in the output cache is retrieved from another source. The contents of the page are then sent to the client computing system.

BRIEF DESCRIPTION OF THE DRAWINGS

[13] The present invention is illustrated by way of example and not limitation in the accompanying figures in which like reference numerals indicate similar elements and in which:

[14] FIGURE 1 illustrates an example of a conventional computing system;

[15] FIGURE 2 shows an embodiment of the invention in which a client computing system and a web server are communicating with one another;

[16] FIGURE 3 illustrates the web server of FIGURE 2 in more detail;

[17] FIGURES 4A and 4B are a flowchart that describes the processing in a server computing system in an embodiment of the invention;

[18] FIGURE 5 is a flowchart for explaining the processing that occurs during a rendering process;

[19] FIGURE 6 is a flowchart for explaining the processing that occurs when a cache timer expires in an embodiment of the invention;

[20] FIGURES 7 and 9 are exemplary dynamic content files; and

[21] FIGURES 8, 10A and 10B are exemplary page control files referenced by the dynamic content files of FIGURES 7 and 9, respectively.

DETAILED DESCRIPTION OF THE INVENTION

[22] Embodiments of the present invention provide a framework, whereby portions of a web page may be stored, for a limited period of time, in an output cache associated with a web server. If the portion is required for a requested web page and the portion is stored in the output cache, then the portion will be injected into the markup language, for example, HTML, or any other authoring language supported by a web browser, for delivery to the browser.

[23] FIGURE 1 is a schematic diagram of a conventional general-purpose digital computing environment that can be used to implement various aspects of the invention. Computer 100 includes a processing unit 110, a system memory 120 and a system bus 130 that couples various system components including the system memory to the processing unit 110. System bus 130 may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures. System memory 120 includes a read only memory (ROM) 140 and a random access memory (RAM) 150.

[24] A basic input/output system (BIOS) 160 containing the basic routines that help to transfer information between elements within the computer 100, such as during start-up, is stored in ROM 140. Computer 100 also includes a hard disk drive 170 for

reading from and writing to a hard disk (not shown), a magnetic disk drive 180 for reading from or writing to a removable magnetic disk 190, and an optical disk drive 191 for reading from or writing to a removable optical disk 192, such as a CD ROM or other optical media. Hard disk drive 170, magnetic disk drive 180, and optical disk drive 191 are respectively connected to the system bus 130 by a hard disk drive interface 192, a magnetic disk drive interface 193, and an optical disk drive interface 194. The drives and their associated computer-readable media provide nonvolatile storage of computer readable instructions, data structures, program modules and other data for personal computer 100. It will be appreciated by those skilled in the art that other types of computer readable media which can store data that is accessible by a computer, such as magnetic cassettes, flash memory cards, digital video disks, Bernoulli cartridges, random access memories (RAMs), read only memories (ROMs), and the like, may also be used in the exemplary operating environment.

[25] A number of program modules can be stored on the hard disk, magnetic disk 190, optical disk 192, ROM 140 or RAM 150, including an operating system 195, one or more application programs 196, other program modules 197, and program data 198. A user can enter commands and information into computer 100 through input devices, such as a keyboard 101 and a pointing device 102. Other input devices (not shown) may include a microphone, joystick, game pad, satellite dish, scanner, or the like. These and other input devices are often connected to the processing unit 110 through a serial port interface 106 that is coupled to the system bus, but may be connected by other interfaces, such as a parallel port, a game port or a universal serial bus (USB). A monitor 107 or other type of display device is also connected to system bus 130 via an interface, such as a video adapter 108. In addition to the monitor, personal computers typically include other peripheral output devices (not shown), such as speakers and printers.

[26] Computer 100 can operate in a networked environment using logical connections to one or more remote computers, such as a remote computer 109. Remote computer

109 can be a server, a router, a network PC, a peer device or other common network node, and typically includes many or all of the elements described above relative to computer 100, although only a memory storage device 111 has been illustrated in FIGURE 1. The logical connections depicted in FIGURE 1 include a local area network (LAN) 112 and a wide area network (WAN) 113. Such networking environments are commonplace in offices, enterprise-wide computer networks, intranets and the Internet.

[27] When used in a LAN networking environment, computer 100 is connected to local network 112 through a network interface or adapter 114. When used in a WAN networking environment, personal computer 100 typically includes a modem 115 or other means for establishing a communications over wide area network 113, such as the Internet. Modem 115, which may be internal or external, is connected to system bus 130 via serial port interface 106. In a networked environment, program modules depicted relative to personal computer 100, or portions thereof, may be stored in the remote memory storage device.

[28] FIGURE 2 illustrates a web server 206 for dynamically generating web page content for display on a client in an embodiment of the present invention. A client 202 executes a browser 302 that displays a web page 306 on a display device of the client 202. The client 202 includes a client computer system having a display device, such as a video monitor (not shown). An "INTERNET EXPLORER" browser, marketed by Microsoft Corporation, is an example of a browser 302 in an exemplary embodiment of the present invention. Other exemplary browsers include without limitation "NETSCAPE NAVIGATOR" and "MOSAIC" browsers. The browser 302 receives Hypertext Markup Language (HTML) code in a HyperText Transfer Protocol (HTTP) response 310 from a web server 206 and displays the web page as described by the HTML code. Although HTML is described with reference to one embodiment, other authoring languages, including without limitation SGML (Standard Generalized

Markup Language) and XML (eXtensible Markup Language), are contemplated within the scope of the present invention.

[29] The communications between the client 202 and the web server 206 are conducted using a sequence of HTTP requests 308 and HTTP responses 310. Although HTTP is described with reference to the exemplary embodiment, other transport protocols, including without limitation HTTPS and S-HTTP, are contemplated within the scope of the present invention. On the web server 206, an HTTP pipeline 312 receives an HTTP request 308, resolves the URL (Universal Resource Locator) contained in the request, and invokes an appropriate handler for processing the request. The HTTP pipeline then passes the request to a page framework 314 for satisfying the HTTP request.

[30] FIGURE 3 provides a more detailed view of the page framework 314. The HTTP Request is passed to a page factory module 308 associated with an ASP.NET page 311. The page factory module 308 is invoked to handle instantiation and configuration of objects from the ASP.NET page 311. The ASP.NET page 311 is identified or referenced by a unique URL and may be further identified by ".aspx" suffix, although other suffixes may be used. When a request for a particular ".aspx" resource is first received by the page factory module 308, the page factory module 308 searches the file system for the appropriate resource or file (e.g., the .aspx page 310). The file may contain text (e.g., authoring language data) or another data format (e.g., byte-code data or encoded data) that may later be interpreted or accessed by the server to service the request. If the physical file exists, the page factory module 308 opens the file and reads the file into memory. If the requested aspx file cannot be found, the page factory module 308 returns an appropriate "file not found" error message, e.g., by sending an HTTP "404" message back to the client.

[31] Once the ASP.NET page 311 is read into memory, the page factory module 308 processes the file contents to create a page object 313, corresponding to web page

306, which is made up of one or more components, for example, components 1 through 3, which correspond to controls, as further described below. When the component corresponds to a user control that has associated output available in the output cache, then the output is retrieved from the output cache and injected into the web page to be sent to the browser.

[32] FIGURE 4A and 4B show a flowchart that illustrates the processing in the web server for satisfying a request from a browser.

[33] At P400, the web server receives a request from a browser for a URL. The HTTP pipeline 312 resolves the URL request and passes the request to the page factory module.

[34] At P402, the page factory module determines whether the requested web page has already been parsed or compiled. If the web page was not parsed or compiled, then it is parsed and compiled at P404. Parsing and compiling creates a class that extends a page class.

[35] At P408, the page factory module 308 references the first control of the page. The controls will be described in more detail later.

[36] At P410, a check is made to determine whether a control is a user control that supports output caching. As described later, output caching support can be determined by the presence or absence of an output cache directive.

[37] If output caching is not supported, then, at P418, a component corresponding to the user control is created.

[38] If the check at P410 determines that the control is a user control that supports output caching, then a cache key is determined and at P414, a determination is made as to whether a cached result corresponding to the user control is in the cache. The cache

key is a Globally Unique Identifier (GUID) that is unique for each occurrence of a user control on the page. The cache key is created at compile time.

[39] If the result is not in the cache, then P418 is performed, as before, to create a component corresponding to the control.

[40] If the result is in the cache, then at P415, the cached result is retrieved and a component corresponding to the cached result is created.

[41] At P416, the component is inserted into a tree, thereby creating a hierarchical data model having linked components. That is, each component is linked to at least a prior component and a next component, assuming a prior or a next component exists.

[42] At P424, a check is made to determine whether the last control of the requested page had been examined. If the last control had not yet been examined, then P426 is performed to point to the next control and P410 is again performed to determine whether the next control is a user control that supports output page caching.

[43] If the check at P424 determines that the last control had been examined then, at P428, the page is rendered and at P430 the contents of the page is transmitted to the client computing system. The page is rendered using the appropriate markup language, such as HTML for a web browser or any other web page authoring language supported by the web browser.

[44] FIGURE 5 is a flowchart which expands upon and illustrates, in more detail, the processing that occurs in the exemplary embodiment when rendering the page at P428.

[45] At P502, the first component of the tree is retrieved and at P510 the component is rendered into HTML, for example, or any other web page authoring language supported by the browser.

[46] At P520, a check is made to determine whether the rendered component is a user control that supports output caching based on an output cache directive contained in the component. If output caching is supported, then at P524, the results are saved to the output cache and at P526 a timer is set to an amount of time, for example, in seconds, specified in the output cache directive for output caching the component.

[47] At P522, the tree is checked to determine whether any more components exist. If so, then at P528, the next component is obtained and P510 will again be performed to render the component into a web page authoring language supported by the browser. Otherwise, if the check at P522 determines that no additional components exist in the tree, then rendering of the page is complete.

[48] FIGURE 6 is a flowchart that explains the processing that occurs when the output cache timer, which was set at P526, expires. At P600, the output cache entry corresponding to the expired timer is purged from the output cache, thereby making the cached results unavailable.

[49] FIGURE 7 illustrates an exemplary hierarchical dynamic content file 700, for an exemplary web page in an embodiment of the present invention. In the illustrated embodiment, file 700 contains plain-text declarations in an exemplary dynamic content file format using the ASP.NET framework provided by Microsoft Corporation of Redmond, Washington. The ASP.NET framework allows developers to create "ASP.NET" web page files that typically include C#, Visual Basic or Jscript code, as well as other HTML code. The ASP.NET file contains declarations, or tags, that perform various functions, as well as VB, C# or Jscript code. These declarations are generally easier to write than writing actual programming code.

[50] The particular embodiment of file 700 is accessed when the URL, for example, http://www.microsoft.com/page.aspx, is resolved. The first line of the file 700 provides a registration for a user control via a "REGISTER" directive. The "REGISTER" directive has the following format:

<% Register TagPrefix="*TagPrefixName*" TagName="*Name*" src="*fileIdentifier*" %>

[51] The TagPrefix and TagName are used when the user control is referenced later in file 700. In this particular embodiment, the user control is referenced in line 11 of file 700, in which an optional ID, "UserCon1" is used. The optional ID allows a programmatic reference to be made to a particular occurrence of the user control. The "src" tag refers to a unique identifier where the web server 316 may find the source file for the user control. Each user control used within a file typically possesses a Register directive to permit the ASP.NET server to resolve all references to user controls used in the file.

[52] Lines 2 through 7 of file 700 comprise a code declaration block. Generally, code declaration blocks define user objects and control object member variables and methods that are executed on the server. Thus, the code declaration block in this example is a server control because it defines code to be executed on the server. The first line of the code declaration block is in the format:

[53] <script [language = "*language*"] runat = "server" >

.

.

.

</script>

where the language parameter is optional and the parameters may appear in any order. In the present exemplary embodiment of the present invention, code declaration blocks are defined using <script> tags that contain a "runat" attribute having a value set to "server". Optionally, a "language" attribute may be used to specify the syntax of the inner code. In this example, the language attribute indicates that the code declaration block is written in a language called C#. It should be understood that the disclosed syntax is used in one embodiment of the present invention, however,

alternative embodiments may employ different syntaxes within the scope of the present invention.

[54] Line 11 of file 700 references the user control that was registered in line 1 of file 700. As one can see, the reference to the user control in line 11 uses a tag prefix of "Fragment" and a tag name of "simple", as specified by the Register directive on line 1 of file 700. The user control, in this example, is defined in a file called "fragment.ascx", as specified by the Register directive of line 1 of file 700.

[55] The following, with reference to FIGURES 4A, 4B and 5, explains the processing within a web server, in the exemplary embodiment of the invention, when a request for a URL, corresponding to file 700, is made from a client computer system having a web browser.

[56] First, a user, via the web browser, requests, for example, the URL http://www.microsoft.com/page.aspx. At P400, the request for the URL is received by the web server. The HTTP pipeline within the web server attempts to resolve the URL request and passes the request to the page factory module which causes the file 700 to be accessed.

[57] At P402, a check is made to determine whether the requested page was parsed or compiled, and if not, the page will be parsed and compiled at P404.

[58] At P408, the file 700 is examined for references to any controls. The Register directive, on line 1 of file 700, indicates the existence of a user control that is defined in a file called "fragment.ascx" which will be referenced by a tag prefix of "Fragment" and a tag name of "simple". In this example, no other user controls are present; however, as mentioned earlier, the code declaration block of lines 2 through 7 define a server control and the code at lines 13 through 14, specifically, "<B id=CreatedStamp runat=server></B>" defines another server control.

[59] At P408, the first control on the page, lines 2 through 7 of file 700 is found.

[60] At P410, the control is determined not to be a user control that supports output caching. Therefore, P418 is performed to create a component corresponding to the control.

[61] At P416, the component is added to a tree, thereby building the data model.

[62] At P424, a check is made to determine whether there are other controls in the page. In this example, there are other controls. Therefore, P426 is performed to get the next control.

[63] The next control, in this example is a user control referenced by file 700 and determined to be on line 11 of file 700. In this example, this is the only user control referenced by file 700.

[64] At P410, a determination is made as to whether the user control supports output caching. FIGURE 8 shows file 800, which is the file "fragment.ascx". Line 2 of file 800 is an output cache directive, indicating that the output produced by file 800 is to be cached for a time period of 10 seconds. Therefore, the user control does support output caching.

[65] At P412, the output cache key, corresponding to a GUID is determined. The GUID was created at compile time.

[66] At P414 a determination is made as to whether the cached result is available in the output cache.

[67] If the cached result is not available, then at P418, a component is created that corresponds to the user control. In this example, the component includes the content of file 800, which includes instructions for obtaining information encapsulated with instructions for displaying the information on the client computing system's browser.

[68] If, at P414, it is determined that the cached result is available, then the cached result is obtained and stored into a component.

[69] At P416, the cached result is inserted into the tree, thereby further building the data model.

[70] At P424, file 700 is examined to determine whether any other controls exist in the page. A server control is found at lines 13 through 14, as mentioned above. Therefore, P426, P410, P418 and P416 are performed as described above with reference to the first control.

[71] At P424, files 700 and 800 are examined to determine if there are any other controls. Since no other controls exist, in this example, the page is rendered at P428 and the rendered page is transmitted to the client computing system's browser.

[72] The flowchart in FIGURE 5 expands upon the processing of P428 in more detail.

[73] At P502, the first component of the tree is examined. In this example, there is three components in the tree. Only the second component corresponds to a user control that supports output caching.

[74] At P502, the first component of the tree is examined.

[75] At P510, the component is rendered.

[76] At P520, the component is determined not to be a component that corresponds to a user control that supports output caching. Therefore, P522 will be performed next.

[77] At P522, it is determined that more components exist. The next component will be retrieved at P522 and P510 will be performed to render the component.

[78] At P520, the component is determined to correspond to a user control component that supports oputput caching. If, the cached result was available at P414, then the cached

result was stored in the component at P415. If this is the case, then the cached result is rendered, at P510, and the check at P520 would conclude that the component does not support output caching, because the contents of the component were already cached.

[79] If the second component of the tree, in our example, refers to a user control with results that were not available in the output cache at P414, then the contents of file 800, i.e., the contents of the user control, are included in the component. At P510, the contents of the component are rendered and at P520, a determination is made that the component does support output caching. The rendered component is then saved in the output cache, at P524, and at P526, a timer is started to time the amount of seconds that the cached entry is allowed to be stored in the output cache.

[80] At P522, a determination is made that a third component exists and P528, P510 and P520 will be performed in the same manner as was done for the first component

[81] At P522, a determination is made that no additional components in the tree exist and the rendering process is complete.

[82] FIGURE 8 shows the exemplary contents 800 of the exemplary user control referenced by line 11 of file 700. Line 1 is a language directive that indicates the language used. In this example, line 1 indicates that the user control is written in C#. Line 2 is an Output Cache Directive that indicates that the output fragment produced by executing the instructions within user control 800 is to be stored in an output cache for a duration of 10 seconds.

[83] Referring back to file 700 in FIGURE 7, the instruction on line 4 sets a variable, "DateTime Created" to the present date and time, i.e., "DateTime.Now". At line 5, the date is formatted and the resulting string is set into the CreatedStamp tag. Lines 7-13 contain a code declaration block which references the user control causing the user control to be executed if the output generated by the user control is not already

available in the output cache. If the user control is available in the output cache, the output of the user control is retrieved from the output cache. Thus, in the situation in which the referenced user control is available in the cache because, for example, the URL http://www.microsoft.com/page.aspx was requested, for example, 5 seconds earlier, the following text will be sent to the client computing system.

Fragment Cache created: May 22, 2001, 2:05:40 PM

This Page was created at May 22, 2001, 2:05:45 PM

[84] FIGURE 9 shows another exemplary hierarchical dynamic content file 900 in an embodiment of the present invention. Like file 700, file 900 also contains plain-text declarations in an exemplary dynamic content file format using the ASP.NET framework provided by Microsoft Corporation. This example is similar to that of FIGURE 7, but differs from FIGURE 7 in that the user control, which appears on line 11 of file 900, uses the "VaryBy" feature. In this particular case, the "VaryByControl" feature is used.

[85] The "VaryByControl" feature identifies one or more controls by name to uniquely identify cache entries. For example, FIGURES 10A and 10B show file 1000, which is an exemplary user control file, fragment.ascx, referenced by file 900 in FIGURE 9, line 11. Line 2 of file 1000 contains an output cache directive, which specifies that the output produced by executing the instructions within file 1000 can be stored in the output cache and made available for a period of 60 seconds. The output cache directive contains the "VaryByControl" feature, indicating the name "Category". "VaryByControl="Category" informs the output cache to vary cached items based on the value of "Category". That is, the output is placed in different cache entries based on the value of "Category".

[86] The instructions of the user control of file 1000 check the value of "Category", and if the value is empty ("") the value of CategoryItem.InnerHtml is set to "Not cached...".

Otherwise, the value of Category.InnerHtml is set to "You selected: " + Category.Value+. See lines 5-10 of file 1000. At lines 14-17 the current date and time and the date and time at expiration of the caching time duration is determined and stored. Lines 21-37 correspond to an HTML select feature. The HTML select feature, when executed on a browser, opens a drop down list box to be displayed on the browser so that the user may select one of the listed items.

[87] Another feature that can appear in the output cache directive is "VaryByParam". The "VaryByParam" feature is similar to the "VaryByControl" feature, but indicates that output caching is to occur according to the values of the parameters listed with the "VaryByParam feature. The following examples help to clarify this feature. Further, if the "VaryByParam" feature is used with the setting "*", caching is varied by every key/value permutation in a query string or by values that are sent from a web browser when using a <form> with method equal to HTTP POST or GET.

[88] For example, suppose the following output cache directive appears in the user control file which is referenced by a file specified by URL http://localhost/caching/cache.aspx:

<%@ OutputCache Duration="60" VaryByParam="*"%>

[89] Consider the following example of four different exemplary requests for cache.aspx using the HTTP GET protocol to pass parameters:

1. http://localhost/caching/cache.aspx?count=10&location=dallas
2. http://localhost/caching/cache.aspx?count=10&location=newyork&shipcode=t56
3. http://localhost/caching/cache.aspx?location=dallas
4. http://localhost/caching/cache.aspx?location=seattle

[90] For the above four requests, four corresponding cache entries will exist because no two requests have the same parameters.

[91] If the output cache directive is replaced by:
<%@ OutputCache Duration="60" VaryByParam="location"%>
the output cache is instructed to cache documents that vary by the location parameter. Using the exemplary four requests, requests 1 and 3 will be served by one cache entry, if available, because location = dallas for these two entries.

[92] If the output cache directive is replaced by:
<%@ OutputCache Duration="60" VaryByParam="location;count"%>
the output cache is instructed to cache documents that vary by the location and the count parameter. Using the exemplary four requests, one can see that all four requests would be cached separately because requests 1 and 3 vary by count.

[93] Another feature that can be used with the output cache directive is "VaryByCustom", which can have values, VaryByCustom="browser", or VaryByCustom="[user defined string]". For example, if the following output directive is used:

<%@ OutputCache Duration="60" VaryByCustom="browser"%>

the output will be cached according to the type of browser requesting the information. That is, the cache will vary by browser name and major browser version.

[94] In order to provide monitoring capability to the web server to enable web page developers to observe the health and performance of caching and of applications running on the server, performance and monitoring counters may be published by, for example, the ASP.NET framework via an Application Program Interface (API), for example, the standard Windows NT PerfMon Application Program Interface (API). This information can be accessed via a "Performance Monitoring" administration tool

which is available on all Windows NT systems from Microsoft Corporation of Redmond, Washington.

[95] ASP.NET supports the following "Application" performance counters, which can be used to count the performance of a single instance of an ASP.NET application. A unique instance appears for these counters named "__Total__" which aggregates these counters for all application instances on a machine. The "__Total__" instance is always available – the counters will be zeroed when no specific application instances are present. These counters will be exposed under the "ASP.NET Applications" performance counter object within PerfMon.

[96] Cache Total Entries

Total number of entries within the cache. This counter includes both internal use of the cache by the ASP.NET framework and external use of the cache through exposed APIs.

[97] Cache Total Hits

Total number of hits from the cache. This counter includes both internal use of the cache by the ASP.NET framework and external use of the cache through exposed APIs.

[98] Cache Total Misses

The number of failed cache requests per application. This counter includes both internal use of the cache by the ASP.NET framework and external use of the cache through exposed APIs.

[99] Cache Total Hit Ratio

Total Hit/Miss ratio for the cache. This counter includes both internal use of the cache by the ASP.NET framework and external use of the cache through exposed APIs.

[100] Cache Total Turnover Rate

The number of additions and removals to the total cache per second. It is useful in helping determine how effectively the cache is being used. If the turnover is large, then the cache is not being used effectively.

[101] Cache API Entries

Total number of entries in the cache, when used via the external APIs (i.e. excluding internal use by the ASP.NET framework).

[102] Cache API Hits

Total number of hits from the cache, when used via the external APIs (i.e. excluding internal use by the ASP.NET framework). Implementation of this counter will require creating a pass-through wrapper around the current APIs to return to external developer code when it requests the Cache object.

[103] Cache API Misses

Total number of failed requests to the cache, when used from the external APIs (i.e. excluding internal use by the ASP.NET framework).

[104] Cache API Hit Ratio

Cache Hit/Miss ratio, when used via the external APIs (i.e. excluding internal use by the ASP.NET framework).

[105] Cache API Turnover Rate

The number of additions and removals to the cache per second, when used via the external APIs (i.e. excluding internal use by the ASP.NET framework). It is useful in helping determine how effectively the cache is being used. If the turnover is large, then the cache is not being used effectively.

[106] Output Cache Entries

Total number of entries in the output cache.

[107] Output Cache Hits

Total number of requests serviced from the output cache.

[108] Output Cache Misses

The number of failed output cache requests per application.

[109] Output Cache Turnover Rate

The number of additions and removals to the output cache per second. It is useful in helping determine how effectively the cache is being used. If the turnover is large, then the cache is not being used effectively.

[110] Output Cache Hit Ratio

Percentage of total requests serviced from the output cache.

[111] Embodiments of the invention may be implemented in hardware, software, or firmware. The firmware may be in a read-only memory and the software may reside on a medium such as a floppy disk or an optical disk.

[112] While the invention has been described with reference to certain illustrated embodiments, the words which have been used herein are words of description, rather than words of limitation. Changes may be made within the purview of the appended claims without departing from the scope and spirit of the invention in its aspects. Although the invention has been described herein with reference to particular structures, acts, and materials, the invention is not to be limited to the particulars disclosed, but rather extends to all equivalent structures, acts, and materials, such as are within the scope of the appended claims.